CloudMonatt: an Architecture for Security Health Monitoring and Attestation of Virtual Machines in Cloud Computing

Tianwei Zhang Ruby B. Lee
Princeton University
{tianweiz, rblee}@princeton.edu

Abstract

Cloud customers need guarantees regarding the security of their virtual machines (VMs), operating within an Infrastructure as a Service (IaaS) cloud system. This is complicated by the customer not knowing where his VM is executing, and on the semantic gap between what the customer wants to know versus what can be measured in the cloud. We present an architecture for monitoring a VM's security health, with the ability to attest this to the customer in an unforgeable manner. We show a concrete implementation of property-based attestation and a full prototype based on the OpenStack open source cloud software.

1. Introduction

Cloud customers are concerned about the security of the virtual machines (VMs) they lease. Recently, researchers have suggested a "security on demand" service model for cloud computing, where secure computing platforms are dynamically provisioned to cloud customers according to their specific security needs [24]. This also enables cloud providers to deploy new secure servers, which may have different security features that customers want, while still running unsecured virtual machines on their existing machines. The availability of secure computing platforms is a necessary but not a sufficient solution to convince cloud customers to move their sensitive data and code to the cloud. Cloud customers need further assurance to convince them that the security measures are indeed deployed, and are working correctly. In this paper we present an end-to-end architecture for both monitoring and attestation of a VM's security properties in an IaaS cloud.

In an IaaS cloud, a customer requests to launch a VM in the cloud system. The cloud provider places the VM in a virtualized cloud server, and allocates a specified amount of physical resources (CPU, memory, disk, etc.) to this VM. The customer is granted remote access to this VM. During the VM's lifetime, the customer would like to know if his VM has good security health. A healthy VM satisfies the security properties the customer requested for his leased VM. For example, if the customer stores sensitive data in the cloud server's storage, a healthy VM enforces *confidentiality* protection of the data from other VMs, or from physical attackers. For another customer with time-critical service needs, a healthy VM means that resources that have been contracted for in the Service Level Agreement (SLA) are always *available* to the VM.

In cloud computing, different customers share the same cloud server, as co-tenants or co-resident VMs. These VMs may belong to competitors, spies, or malicious attackers. The security heath of a VM should take into account the other co-resident VMs, not just the attacks from within his VM (e.g., malware, guest OS root kits, etc.). We call this *outside-VM* and *inside-VM* vulnerabilities, respectively. Past work have shown that the "bad neighbor" VMs are able to steal critical information through side-channel attacks [31, 46], thus compromising the VM's *confidentiality health*. Resource contentions between different VMs on the same server motivate malicious VMs to perform the Resource-Freeing Attack [40], thus compromising the victim VM's *availability health*. Large cloud management software, including the hypervisor, will also have bugs [29], which can be exploited to compromise a VM's security health. Hence, a VM's security health depends on not only the activities inside the VM, but also the VM's interactions with the environment.

Monitoring the VMs' security health poses a series of challenges in a cloud system. First, the customer's limited privileges prevent him from collecting comprehensive security measurements to monitor his VM's health securely. He only has access to the VM, but not to the host server. For inside-VM vulnerabilities, once the VM's OS is compromised by the attacker, the customer may not get correct measurements. For outside-VM vulnerabilities, the customer cannot collect information about the co-resident VMs, hypervisor, etc. Second, the customer's desired security requirements are expressed in terms of a VM, but the security measurements usually involve the physical server, the hypervisor and other entities related to this VM. This creates a semantic gap between what the customers want to monitor and the type of measurements that can be collected. Third, the VMs go through different lifecycle stages and may migrate to different host servers. A seamless monitoring mechanism throughout the VMs' lifetime is therefore highly desirable. Fourth, there are numerous entities between the customers and the point of VM operations. It is

ISCA'15, June 13-17, 2015, Portland, OR USA

http://dx.doi.org/10.1145/2749469.2750422

important to collect, filter and process the attestation information securely to attest, i.e., pass on to the customer in an unforgeable way, only the requested information.

In this paper, we design a flexible architecture called *CloudMonatt*, to monitor the security health of customers' VMs within a cloud system. *CloudMonatt* is built upon the *property-based* attestation model, and provides several novel features. First, it provides a framework for monitoring different aspects of security health. Second, it shows how to interpret and map actual measurements collected to security properties that can be understood by the customer. These bridge the semantic gap between requested VM properties and the platform measurements for security health. Third, to the best of our knowledge, this is the first concrete realization of property-based attestation for a VM. Previous work discuss the desirability of property-based attestation, versus binary attestation, but did not give any implementations. Fourth, attestations can be done at runtime and for VM migrations, not just at boot up and VM launch time. Fifth, *CloudMonatt* provides remediation response strategies based on the monitored results.

Key contributions in this paper are:

- Definition of "security health" of a VM for several different security properties.
- Design of a flexible architecture to monitor the security health of VMs on cloud servers over the VMs' lifecycle.
- Concrete examples to show how to bridge the semantic gap between security properties and measurements.
- Providing different security monitoring and attestation activities during aVM's lifecycle.
- Providing automatic remediation responses to failing security health indicated by negative attestation results.
- Full prototype of the architecture with property-based attestation in a cloud infrastructure.

Section 2 reviews the background and related work. Section 3 describes the *CloudMonatt* architecture and its essential monitoring and attestation protocols. Section 4 gives concrete examples of security property measurements and their interpretation. Section 5 shows the security monitoring at different VM stages, and the corresponding remediation response strategies. Section 6 gives the details of our prototype implementation. Section 7 shows the performance and security evaluations. We conclude in Section 8.

2. Background and Related Work

Different techniques have been proposed for security monitoring and attestation of VMs. We describe some past work in Virtual Machine Introspection (VMI) and Remote Attestation.

2.1. Virtual Machine Introspection

Past work on *inside-VM* threats proposed Virtual Machine Introspection techniques. This can provide the service of VM health monitoring at the hypervisor level. Since the hypervisor monitor is outside the VM, it is able to detect the existence of malicious or untrusted entities inside the VM, while being isolated, and thus protected, from the VM.

Since the introduction of the VMI technique and the Livewire intrusion detection system [19], many VMI-based architectures have been designed to monitor the inside-VM health, e.g., VMwatcher [25], Ether [13], Lares [28], virtuoso [14], VMST [17], etc. These architectures detect abnormal behaviors inside the VM, but do not consider the threats from co-resident VMs or other outside-VM entities. For instance, a VMI tool may be able to detect confidentiality breaches caused by malicious programs residing in the target VM, but it cannot detect information leakage via cross-VM side channels (as we do, in a concrete example in Section 4). Also, how to use these techniques in the cloud system and allow the remote customer to use these monitoring services are problems which have not been addressed. We address these problems and show how VMI technologies can be seamlessly deployed in our *CloudMonatt* architecture.

2.2. Remote Attestation

Remote attestation has been defined to enable remote customers to test the integrity of a targeted system based on the integrity hash measurements supplied by that system.

TPM-based attestation, proposed by the Trusted Computing Group (TCG) [20, 21], can verify the platform integrity of a remote server. The targeted server uses the Trusted Platform Module (TPM) to calculate the binary hash values of the platform configurations and send them to the customer. The customer compares these values with reference configurations, possibly via a trusted third party appraiser [33], and determines whether the state of the platform is in the unmodified (good) state. Many systems enabled with remote binary attestation have been designed (e.g., Intel's TXT [1], IMA [33], PRIMA [23], BIND [38], Pioneer [37], TVMM [18], etc.). In the context of virtualization platforms, the virtual Trusted Platform Module (vTPM)[8, 16, 35, 41] was designed to provide the same usage model and services to the VMs as the hardware TPM. Then, remote attestation can be carried out directly between the customers and their virtual machines by the vTPM instances.

vTPM-based attestation raises some problems for VM monitoring: it cannot monitor the security conditions of the VM's environment. Furthermore, the monitoring tool resides in the guest OS, so it needs modification of the guest OS, and commodity OSes are also highly susceptible to attacks.

To overcome the above problems, the concept of *centralized attestation* is introduced in the cloud system to manage the attestation procedure. In [36], Schiffman et al. implemented a centralized "cloud verifier" that can provide the integrity attestations for customers' VM applications. Customers issue the authorization for the VM to access applications only when the integrity attestation passes. In [34], Santos et al. designed a centralized monitor to check the platform's configurations and map them to security attributes. This enables customers'

VMs to be allocated on the platforms with specified attributes. Then Attribute-Based Encryption is exploited to seal and unseal data between customers and cloud servers to ensure they are not compromised. However, [36] and [34] are still based on *TPM-based attestation* for platform integrity and configuration checking, and do not consider other security properties like confidentiality or availability, nor the VMs' interactions (intended or unintended) with the outside-VM environment.

Property-based attestation [32, 30, 12] was proposed, in concept, to attest different properties, functions and behaviors of systems. A trusted third party is introduced to transform the platform's security measurements into properties and vice versa, and determine if the platform's condition satisfies a given set of properties. However, the specification and interpretation of properties to be attested remain as challenging, open problems [27]. They make it very difficult for computer architects to convert the concept of *property-based attestation* into real architectures. We solve some of these problems in this paper with concrete examples of how to monitor host machines or VMs to see if different security properties are being enforced or violated, thus providing perhaps the first concrete realization of property-based attestation in cloud computing.

Unlike past work on attestation which focus on binary attestation of platform integrity, we focus on an infrastructure for property-based attestation of arbitrary security properties, not just integrity. We show concrete examples of the violation of different security properties, like degraded availability and loss of confidentiality through covert channels. We enable attestation not only on boot up and VM initiation, but also during VM runtime and migration. We also propose a novel ongoing periodic attestation for a VM's security health, and automated remediation responses for negative attestation results.

3. CloudMonatt Architecture

3.1. Goals of the Architecture

The goals of the *CloudMonatt* architecture are:

1. To provide a flexible distributed cloud architecture that can detect and monitor the security health of the customers' VM in the cloud, e.g., by detecting its vulnerabilities, the vulnerabilities of the platform it is running on, or the vulnerabilities due to co-resident VMs;
2. To provide a secure protocol to request and receive security property monitoring measurements from the cloud's secure servers, and produce an unforgeable attestation report; and
3. To interpret security health measurements, determine if a requested security property is held for the VM, and enable different remediation responses when the VM's security health is appraised as inadequate.

In this section, we describe the main architecture for achieving goals (1) and (2), which are independent of the specific security properties a server can implement within the *CloudMonatt* architecture. Section 3.2 describes the main architectural components. Section 3.3 describes the threat model, referring to these components. Section 3.4 describes the monitoring and attestation protocols. Goal (3) depends on the specific security property being monitored, and Section 4 gives several concrete examples of property interpretations.

3.2. Architecture Overview

Figure 1 shows an overview of the *CloudMonatt* architecture. This includes four entities: 1) Cloud Customer, 2) Cloud Controller, 3) Attestation Server and 4) Cloud Server.

3.2.1. Cloud Customer: The customer is the initiator and end-verifier in the system. He places a request for leasing VMs with specific resource requirements and security requests to the Cloud Controller. He can issue any number of security attestation requests during his VM's lifetime. Table 1 shows the attestation and monitoring APIs provided to the customers. *CloudMonatt* allows customers to invoke the monitoring and attestation requests at any time during the VM's lifecycle. It also gives the customers two modes of operation: one-time attestation and periodic attestation.

One-time attestation: the customer can request the attestation at any time. Then the Attestation Server performs the required attestation and sends back the results.

Periodic attestation: the customer can ask for periodic attestations with specified constant or random frequency. The cloud server supplies the measurements, and the Attestation Server accumulates and interprets the measurements periodically. The customer receives fresh results periodically and can stop the process at any time.

3.2.2. Cloud Controller: The Cloud Controller acts as the cloud manager, responsible for taking VM requests and servicing them for each customer. The *Policy Validation Module* in the Controller selects qualified servers for customers' requested VMs. These servers need to both satisfy the VMs' demanded physical resources, as well as support the requested security properties and their property monitoring services. The *Deployment Module* allocates each VM on the selected server.

During the VMs' lifecycle, the customers may request the Cloud Controller to monitor the security properties associated with their VMs. The Cloud Controller will entrust the Attestation Server to collect the monitored security measurements from the correct VMs, and send a report back to it. It then sends the results back to the customers to keep them informed of the VMs' security health. When these results reveal potential vulnerabilities for the VMs, the *Response Module* in the Controller carries out appropriate remediation responses.

3.2.3. Attestation Server: The Attestation Server acts as the attestation requester and appraiser, and consists of two essential modules. 1) The *Property Interpretation Module* is responsible for validating measurements, interpreting properties and making attestation decisions. It needs a certificate from a *privacy Certificate Authority* (pCA) to authenticate cloud servers. The *privacy Certificate Authority* may be a separate trusted server already used by the cloud provider for standard certification of public-key certificates that bind a public key

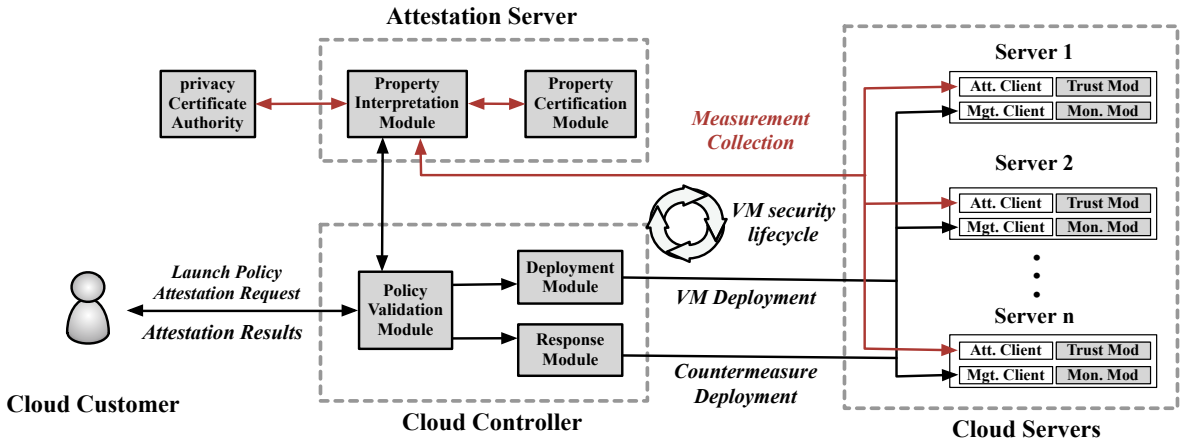

Figure 1: Architectural Overview of *CloudMonatt*

Table 1: Types of Monitoring and Attestation Requests (nonces N are added for freshness for each request)

Request API	Description
startup_attest_current(**Vid**, **P**, **N**)	Invoke an attestation of VM **Vid** for security property **P**, before launching the VM
runtime_attest_current(**Vid**, **P**, **N**)	Invoke an immediate attestation of VM **Vid** for security property **P**
runtime_attest_periodic(**Vid**, **P**, **freq**, **N**))	Invoke a periodic attestation of VM **Vid** for security property **P** at the frequency of **freq** or at random intervals
stop_attest_periodic(**Vid**, **P**, **N**))	Stop a periodic attestation of VM **Vid** for security property **P**

to a given machine. 2) The *Property Certification Module* is responsible for issuing an attestation certificate for the properties monitored. There can be different Attestation Servers for different clusters of cloud servers, enabling scalability of the *CloudMonatt* architecture.

We introduce the Attestation Server for security monitoring/attestation while the Cloud Controller is responsible for management. This job split achieves better scalability, since attestation servers can be added to handle more cloud servers. It consolidates property interpretation in the attestation servers, rather than replicating this in each cloud server, or burdening the Cloud Controller. This also achieves better "separation of duties" security, since the Cloud Controller need only focus on cloud management while the Attestation Server focuses on security. It also improves performance by preventing a bottleneck at the Cloud Controller if it had to handle management as well as myriad attestation requests and security property interpretations.

3.2.4. Cloud Server: The Cloud Server is the computer that runs the Virtual Machine (VM) in question. It is the attester in the system. It provides different measurements for different security properties. Figure 2 shows the structure of a cloud server with a Type-I hypervisor (e.g., Xen [7]). This has the hypervisor sitting on bare metal, and a privileged VM called the host VM (or Dom0) running over the hypervisor. Not all the cloud servers in the cloud provider's data center have to be trusted (almost all existing ones are not), only those servers on which security monitoring is necessary need to be secure. To support *CloudMonatt*'s goals, a cloud server must include a *Monitor Module* and a *Trust Module*.

The ***Monitor Module*** contains different types of monitors to provide comprehensive and rich security measurements. These monitors can be software modules or existing hardware mechanisms like performance counters or the TPM chip. For example, the hardware performance monitor unit (present ubiquitously in Intel x86 and ARM processors) has numerous hardware performance counters to collect runtime measurements of the VMs' activities. An Integrity Measurement Unit (which could use a TPM [20] chip) can be used to measure accumulated hashes of the system's code and static data configuration. In the hypervisor, a VMI introspection tool (examples given in Section 2.1) can be used to collect the information inside the specified VM, and the VMM profile tool can be used to collect dynamic information about each VM's activities.

We define a new hardware ***Trust Module*** in Figure 2. This *Trust Module* is responsible for server authentication using the Identity Key, crypto operations using the Crypto Engine, Key Generation and Random Number generation (RNG) blocks, and secure measurement storage using the Trust Evidence Registers. By using new hardware registers to store the security health measurements (trust evidence), we do not need to include the main DRAM memory in our Trusted Computing Base, although trusted RAM can also be used instead of Trust Evidence Registers in the Trust Module.

Figure 2 also shows the functional steps taken by the Monitor Module and the Trust Module. The Cloud Server includes an *Attestation Client* in the host VM that ① takes requests from the Attestation Server to collect a set of measurements. It invokes the *Monitor Module* to ② collect the measurements and the *Trust Module* to ③ generate a new attestation key for this attestation session. This new attestation key is signed by the *Trust Module*'s private identity key. The required measurements of suspicious events or evidence of trustworthy operation are ④ collected from the *Monitor Module* and ⑤ stored into new *Trust Evidence Registers*. These *Trust Evidence Registers* are analogous to the performance counters

used for evaluating the system's performance, except that they measure aspects of the system's security. The *Trust Module* then ⑥ invokes its *Crypto Engine* to sign these measurements and ⑦ forwards the data to the *Attestation Client* which ⑧ sends it to the Attestation Server. The *Trust Module* contains a *Key Generator* and a *Random Number Generator* for generating keys and nonces.

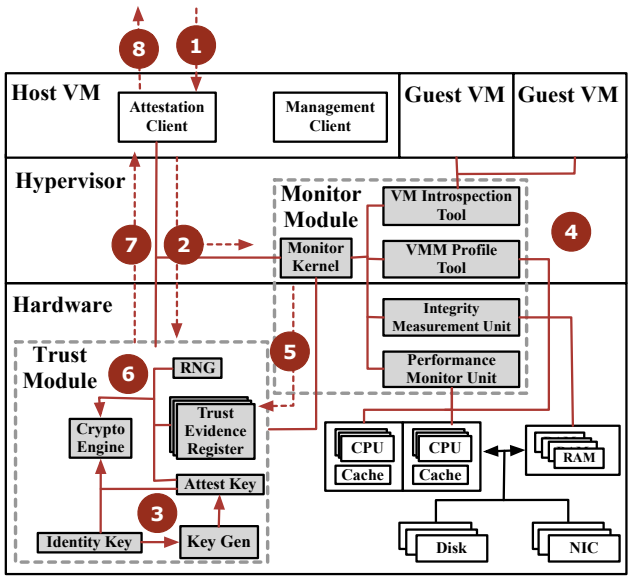

Figure 2: Server Architectures Enabling Security Monitoring include new trusted hardware and software features (shown in grey) in a Trust Module and a Monitor Module.

3.3. Threat Model

The threat model is that of hostile VMs running in the cloud on the same cloud server, or hostile applications or services running inside a VM, that try to breach the confidentiality or integrity of a victim VM's data or code. They may also try to breach its availability, in spite of the cloud provider having allocated the VM its requested resources. The cloud provider is assumed to be trusted (with its reputation at stake), but may have vulnerabilities in the system. We assume that the Cloud Controller and the Attestation Server are trusted - they are correctly implemented, with secure bootup and are protected during runtime. However the Cloud Servers need not be trusted, except for the *Trust Module* and *Monitor Module* in each server. Note that the trusted servers, the Cloud Controller and Attestation Server, can be redundancy protected for reliability and security, and are only a small percent of all the servers in the cloud's data center. Also, not all the thousands of cloud servers need to be CloudMonatt-secure servers.

We focus on two types of adversary's capabilities: (1) An adversary, who tries to exploit vulnerabilities in the customers' VMs, either from inside the VM, or from another malicious VM co-resident on the same server. (2) An active adversary who has full control of the network between different servers, as in the standard Dolev-Yao threat model [15]. The adversary is able to eavesdrop as well as falsify the attestation messages, trying to make the customer receive a forged attestation report without detecting anything suspicious. With regard to this second adversary, *CloudMonatt* needs secure monitoring and attestation protocols which we define next.

3.4. Secure Monitoring and Attestation Protocols

In a distributed architecture where communication is over untrusted networks, the protocols are an essential part of the security architecture: they establish trust between the customer and the cloud provider, and between different computers in the cloud system. In *CloudMonatt*, an attestation protocol must be unforgeable in spite of the network attacker and the other attackers in the untrusted servers. This requires secure communications among the four entities in Figure 1, and unforgeable signatures of the measurements and the attestation report from the place of collection (in the Cloud Server) through the Attestation Server, Cloud Controller and finally to the customer. We first describe the main attestation protocol. Details of the cryptographic keys involved, the secure communications and storage will be clarified later.

Figure 3 shows the attestation protocol in *CloudMonatt*. Initially the customer sends to the Cloud Controller the attestation requests including the VM identifier **Vid**, the desired security properties **P** and a nonce $\mathbf{N}_1$. The Cloud Controller knows the current mapping of all VMs to their assigned cloud servers, and hence forwards the request to the Attestation Server, after adding cloud server identifier **I** to the attestation request. The Attestation Server then requests security monitoring measurements (**rM**) from the Cloud Server **I** where the VM is running. The Cloud Server collects the required measurements **M**, calculates the quote $\mathbf{Q}_3$ as the hash value of (**Vid**, **rM**, **M** and nonce $\mathbf{N}_3$), and sends these values back to the Attestation Server. (We borrow the term "Quote" from TPM notation, to represent a cumulative hash measurement.) The Attestation Server checks the signature and hash values, interprets the measurements **M** and property **P**, and generates the attestation report **R**. This attestation report is signed by the Attestation Server and transmitted securely to the Controller, and then signed by the Controller and transmitted back to the customer. Three different nonces $\mathbf{N}_1$, $\mathbf{N}_2$ and $\mathbf{N}_3$ are used to prevent replay attacks over the three channels between each successive pair of servers, for each attestation request.

3.4.1. Secure Storage and Communications: For secure storage, the *Trust Module* provides *Trust Evidence Registers* for attestation measurements, which are only accessible to the *Trust Module* and *Monitor Module*. Accesses to the databases in the Cloud Controller and the Attestation Server are also protected to ensure data confidentiality and integrity.

For secure communications over networks, the *CloudMonatt* architecture expects the customer, Cloud Controller, Attestation Server and secure Cloud Servers to implement the SSL protocol. Our contribution is defining the *contents* of

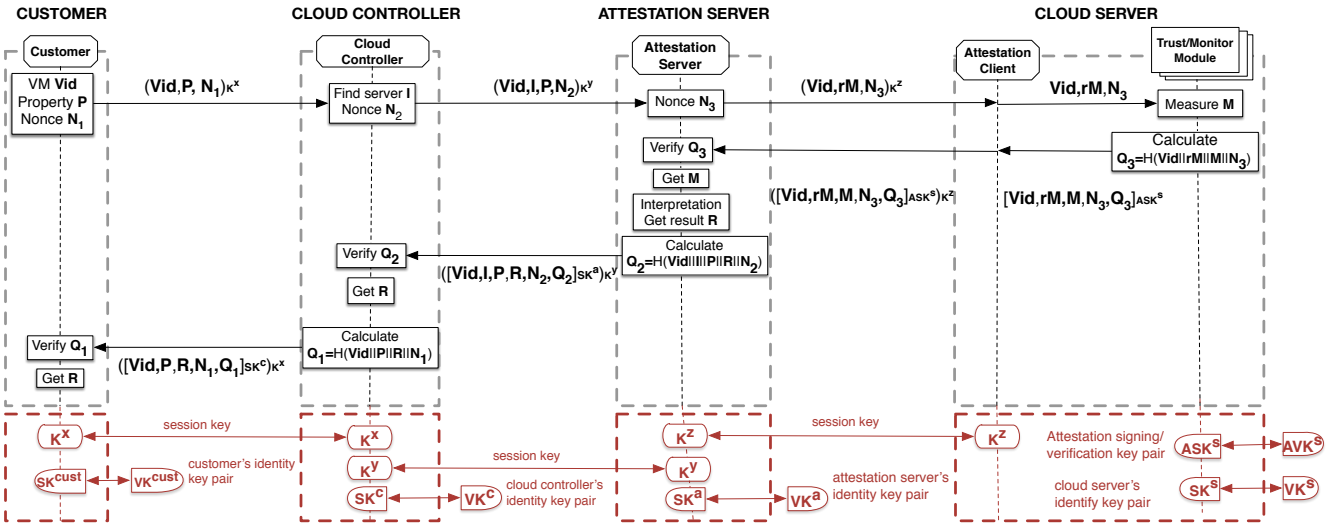

Figure 3: Attestation Protocol and Key Management in *CloudMonatt*. We use the notation $[M]_K$ for a private key operation with key K, $\{M\}_K$ for a public key operation with key K, and $(M)_K$ for a symmetric key operation with symmetric key K.

the SSL messages, and the keys and signatures required for unforgeable attestation reports and Cloud Server anonymity.

3.4.2. Cryptographic Keys Used: We now describe the keys used in Figure 3. The Cloud Controller, Attestation Server and each secure Cloud Server must have one long-term public-private key-pair that uniquely identifies it within the cloud system. This is minimally what is required for SSL support, and is already present in all cloud servers. Hence, each secure cloud server owns a pair of public-private identity keys, $\{\mathbf{VK}^s, \mathbf{SK}^s\}$. The private key, $\mathbf{SK}^s$, can be burned into the *Trust Module* when manufactured, or more preferably, securely inserted into a non-volatile and tamper-proof register in the *Trust Module* when the server is first deployed in the cloud data center. It is never released outside of the *Trust Module*. The public key, $\mathbf{VK}^s$, can be used to authenticate the cloud server. A cloud server mainly uses this identity key-pair to generate a temporary key pair for each attestation request.

A new session-specific key-pair, $\{\mathbf{AVK}^s, \mathbf{ASK}^s\}$, is created by the *Trust Module* whenever an attestation report is needed, so as not to reveal the location of a VM. (An attacker may try to find the server which hosts the victim VM, then he can try to co-locate his VM on the same server. We do not want our attestation protocol to help an attacker do this [31].) The public attestation key $\mathbf{AVK}^s$ is signed by the Cloud Server's $\mathbf{SK}^s$ and sent to the pCA for certification. The pCA verifies the signature via $\mathbf{VK}^s$ and issues the certificate for $\mathbf{AVK}^s$ for that server. This certificate enables the Attestation Server to authenticate the Cloud Server "anonymously" for this attestation.

For secure communications between the servers, SSL first authenticates sender and receiver using their public-private key-pairs, then generates symmetric session keys for encrypting the messages passed between each pair of servers. Hence, Figure 3 shows the communications between the customer and the Controller protected with a symmetric session key $\mathbf{K}^x$, between the Controller and the Attestation Server with a symmetric session key $\mathbf{K}^y$, and between the Attestation Server and Cloud Server with symmetric session key $\mathbf{K}^z$.

In the next section, we elaborate on what security health monitoring means for different security properties like confidentiality and availability, in addition to integrity. In past work, integrity has been the primary, if not the only security property measured (and usually only on bootup). We give concrete examples to illustrate the definition and monitoring of a broader range of security properties, including example attacks, to illustrate potential security breaches in the cloud.

4. Security Health Monitoring

We define the *Security Health* of a Virtual Machine as an indication of the likelihood of its security being affected by the actions of hostile VMs co-resident on the same Cloud server, or hostile applications, services or malware within the VM itself. Different indicators of different aspects of security health can be monitored. In our context, these different aspects of security are the security properties requested by the customer. These security properties can be monitored by the various monitors in the server's *Monitor Module* or collected by the *Trust Evidence Registers* in the server's *Trust Module*. The *CloudMonatt* architecture is flexible and allows the integration of an arbitrary number of security properties and monitoring mechanisms, including logging, auditing and provenance mechanisms.

To monitor and attest a security property, three requirements must be satisfied: (1) the Attestation Server can translate the security property, requested for attestation by the customer, to the measurements to request from the target cloud server; (2) the target cloud server implements a *Monitor Module* that can collect these measurements, and a *Trust Module* with a *Crypto Engine* that can securely hash and sign the measurements and send them back to the Attestation Server. (3) the *Property Interpretation Module* in the Attestation Server is able to verify

the measurements and auxiliary information, and interpret if the security property is satisfied.

4.1. Property Mapping and Interpretation

The Attestation Server has a mapping of security property **P** to measurements **M**. This gives a list of measurements **M** that can indicate the security health with respect to the specified property **P**. The Attestation Server can also behave as the property interpreter and decision maker: when it receives the actual measurements **M'** from the server and VM, it can judge if the customers' requested security properties are being enforced. (A simpler Attestation Server may just pass back the measurements **M'** without performing any interpretation or initiating any remediation responses.)

There are many possible security properties that a customer may want. They may include specific properties related to the cornerstone security properties of Confidentiality, Integrity and Availability. We illustrate below with a few examples to show that *CloudMonatt* is flexible enough to support a variety of detection mechanisms. The detection of abnormal VM behaviors is orthogonal to our work, and new methods can easily be integrated into the *CloudMonatt* framework.

4.2. Case Study I: Startup integrity

We start with the well-known use case supported by TPM [20], where a customer wants to check the integrity of both the host platform and the VM before launching his VM in the cloud.

4.2.1. Example Attacks: Attackers (inside-VM or outside-VM) may try to launch a malicious hypervisor, host OS, or guest OS. These software entities could have been corrupted during storage or network transmission. Similarly, the VM image could have been compromised, with malware inserted.

4.2.2. Monitoring Mechanism: The monitoring mechanism involves accumulated cryptographic hashes of the software that is loaded onto the system, in the order that they are loaded. A standard TPM chip can be used, and integrated into the hardware platform. The measurement is typically done in two phases: First, the server's platform configuration (hypervisor, host OS, etc.) is measured (i.e., hashed) during server bootup. Second, the VM image is measured before VM launch.

The Attestation Server can have full knowledge of the attested software, and the correct pre-calculated hash values of its executable files. It can use these correct values to check the hash measurements sent back by the cloud server, and issue the integrity property attestation, if the hash values match. Alternatively, the Attestation Server can use a trusted Appraiser system (like an Integrity Measurement Architecture (IMA) [33]) to check if the measured hash values conform to the correct values for a pristine, malware-free system, before sending the Startup Integrity Property attestation back to the customer.

4.3. Case Study II: Runtime Integrity

The customer may want to know if his VM is infected with malware during runtime, not just at startup time as with TPM-based attestation.

4.3.1. Example Attacks: The attacker can spread virus into the customer's VM. Then the malware inside-VM can compromise the customer's critical programs. Once the malware gets root privilege in the OS, it can compromise the whole VM.

4.3.2. Monitoring Mechanism: A common technique to monitor the VM's health uses VM Introspection (VMI) [19, 14, 17]. implemented as a hypervisor-level monitor. VMI allows the hypervisor to monitor the VM from outside the VM, and examine the states of the target VM. Different VMI tools have been designed to detect and analyze the malware inside the VMs, such as VMwatcher [25] and Ether [13]. These tools can be integrated into *CloudMonatt*. For example, when customers ask to check if there is malware running as a background service and hiding itself in the target VM, the Attestation Server can issue a request for getting the list of running tasks for that VM. The *VM Introspection Tool* located in the hypervisor's *Monitor Module* can probe into the target VM's memory region to obtain the running tasks list [25]. This information will be written into the *Trust Evidence Registers* and transmitted back to the *Attestation Server*. The customer can compare this actual task list in the returned Attestation Report and compare it with the one he gets from querying the corrupted guest OS, to detect the malware running in his VM.

4.4. Case Study III: Runtime Confidentiality Breach through Covert Channels

For VMs with confidential code or data, cryptography is typically used to protect confidential data-at-rest and data-in-transit. However, during execution, the confidential data is decrypted and any secret key being used is also decrypted. During this time, although VMs are protected (isolated) from each other by the hypervisor, it may still be possible to leak the secret crypto key used via a cross-VM covert channel or side channel.

4.4.1. Example Attacks: Hypervisors enable memory protection by enforcing isolation between VMs. However, covert channels still exist across VMs running on the same server. A covert channel exists when a colluding insider (e.g., a program inside the victim VM) can use a medium not normally used for communications to leak secret information to an unauthorized party in another VM. No security policies are overtly broken by overt communications, but are broken by covert communications. When VMs on the same server share physical resources, the contention for these shared resources can leak information, e.g., in the form of timing features. For side channels, past work have demonstrated the shared cache can be exploited by a hostile VM to extract crypto keys from the victim VM [46, 43, 6, 22, 47]. For covert channels, two VMs can encode and transmit information by generating certain characteristics of the shared hardware, which can be detected in certain cases, as in CC-hunter [11]. For example, different cache operations (hit or miss) [31, 45], or memory bus activities (locked or unlocked bus) [44], may be indicative of

certain side-channel or covert channel operations, and may be detectable with new hardware monitoring features.. To illustrate how easy it is to leak information through a covert channel, we design a new cross-VM covert-channel as a case study. We also show how it can be monitored and detected in *CloudMonatt*.

CPU-based covert-channel attack: The basic idea for this covert channel is to use the CPU as the channel medium to transmit information. The sender VM can occupy the CPU for different amounts of time, to indicate different information (e,g, long CPU usage indicates a "1" while short CPU usage signals a "0"). For example, in Xen, the sender VM can trick the Xen scheduler to achieve a fine control over CPU usage. It first requests two colluding virtual CPUs and keeps them idle for some time to build up Xen scheduling credits. Then the main attacker vCPU can sleep to yield the CPU resource, or wake up by exploiting Inter Processor Interrupts (IPI) from the other vCPU to the main attacker vCPU, to give it high priority with the scheduler. When the sender main vCPU and receiver VM share the same CPU, the receiver VM can measure its own execution time, to infer the sender VM's CPU activity, and thus, infer the covert channel information leak. Figure 4 shows the sender VM's CPU usage, observed by the receiver's VM. This covert channel has a high bandwidth of 200bps.

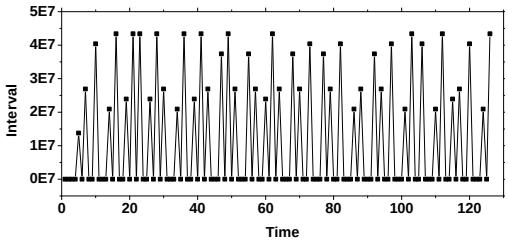

Figure 4: Cross-VM Covert Information Leakage

4.4.2. Monitoring Mechanism: Covert channels are based on contention for shared resources. Programs involved in covert channel communications give unique patterns of the events happening on such hardware [11]. The Attestation Server invokes the *Monitor Module* on the target server to collect the necessary information for real time analysis.

We use CPU usage intervals to detect the existence of the covert channel attack we just created. We set the interval granularity as 1ms. Since the default execution interval in Xen is 30ms, we use 30 programmable *Trust Evidence Registers* to count the occurrence of each CPU usage interval, (0,1],(1,2],...,(29,30], experienced by the sender VM. Suppose the sender VM executes for 4.6ms, then the Trust Evidence Register (4,5] will be incremented by 1. The distribution of CPU usage intervals can reveal the existence of covert channels when the sender VM maliciously changes the time interval to transmit information.

After a certain detection period, the 30 *Trust Evidence Registers* give the distribution of the different CPU usage intervals. These 30 values are sent as the security health measurements for detecting this type of covert channel communications.

4.4.3. Covert-Channel Property Interpretation: When the Attestation Server receives the 30 values, the *Property Interpretation Module* calculates the probability distribution (shown in Figure 5) of the CPU usage intervals. If a covert channel exists, the distribution graph gives two peaks: each peak representing the activity of transmitting a "0" or a "1", respectively. For a benign VM, it typically gives one peak for the default interval of 30 ms. The Attestation Server can use machine learning techniques to cluster the covert-channel results and benign results. (We use 30 bins in our experiment, but a different number can be used to save space or increase accuracy.) This is only one type of covert channel and other types of covert channels can also be monitored (with more Trust Evidence Registers and mechanisms). The system could also be designed to switch randomly between monitoring different sources of covert channels, and use the periodic attestation mode.

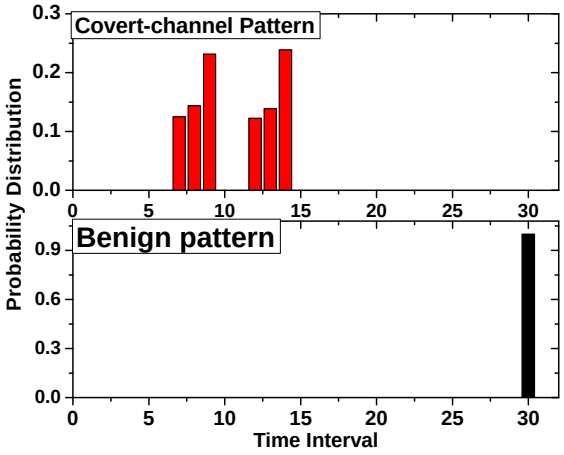

Figure 5: Measurements of Covert-channel Vulnerabilities

4.5. Case Study IV: Runtime CPU Availability

Availability of the resources and services agreed upon by the cloud customer and the cloud provider in the Service Level Agreement (SLA) is a very important security problem in cloud computing. Even if over-provisioning is practiced, the cloud provider is still responsible for providing a fair resource allocation for each VM based on its SLA. During runtime, the customer wants to know if his VM is given the requested resources as paid for. We now show an example of an availability attack, and how CPU resource availability can be monitored.

4.5.1. Example Attacks: An attacker may try to get more resources to severely reduce the availability of shared resources to a victim VM, thus degrading its performance. This may be to improve the attacker's own performance, or it may just be to attack the victim and deny him his rightful use of cloud resources. To achieve this goal, the attacker VM can change

its own workloads to steal more resources from the victim. A typical example is the CPU availability attack against Xen's credit scheduler [48]. The attacker can also change the victim VM's behavior to give up computing resources to the attacker, such as in Resource-Freeing Attacks (RFA) introduced in [40]. For this case study, we demonstrate a new CPU resource availability attack, and use it as an example of resource availability monitoring in *CloudMonatt*.

CPU resource availability attack: This attack targets the boost mechanism of Xen's credit scheduler algorithm [5]. Specifically, each VM receives some credits periodically, and the running VM pays out credits. The Xen scheduler wakes up the VM with extra credits in Round-Robin order. However, when a VM is woken up by certain interrupts, it always gets higher priority to take over the CPU. So the attacker's strategy is to launch a VM with multiple vCPUs and use them to keep sending and receiving Inter Processor Interrupts (IPIs) to each other, so one of the attacker's vCPUs always has the highest priority. Since the attacker's VM always has higher priority than the victim VM, they consume a lot of CPU resources, thus starving the victim's CPU usage.

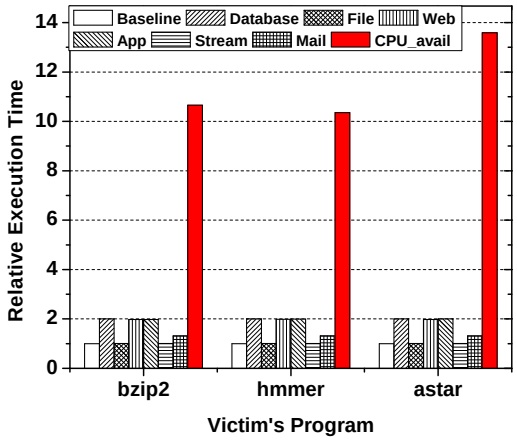

Figure 6: Performance for CPU Availability Attacks.

Figure 6 shows the results for the denial of CPU service attack. The attacker VM and victim VM are located on the same CPU using a Xen hypervisor. The victim VM runs three CPU-bound programs from the SPEC2006 benchmark suite. The attacker VM runs different services typically done in the cloud, as well as the CPU availability attack we designed. When the attacker is I/O-bound (File, Stream or Mail servers), the attacker does not consume much CPU and the victim VM has no performance degradation. When the attacker runs CPU-bound tasks (Database, Web or App servers), the victim's execution time is doubled since it can get a fair share of 50% of the CPU quota. However, when the attacker performs the CPU availability attack described above, the victim's performance is degraded by more than ten times.

4.5.2. Monitoring mechanism: The basic idea for availability monitoring is to measure the resource usage of the attested VM, e.g., CPU usage in this example. During the testing period for CPU availability, the *VMM Profile Tool* measures the attested VM's CPU time: it observes the transitions of each virtual CPU on each physical core, and keeps record of the virtual running time for the attested VM. After the testing period, the *VMM Profile Tool* stops the measurements and calculates the total virtual running time: *CPU_measure*. This measurement is written into one *Trust Evidence Register*, signed and sent back to the Attestation Server.

4.5.3. Availability Property Interpretation: The Attestation Server retrieves the attested VM's virtual running time and calculates the relative CPU usage as the ratio of a VM's virtual running time to real time. If the relative CPU usage is very small, then the Attestation Server interprets the VM's CPU availability as compromised (as shown in Figure 7).

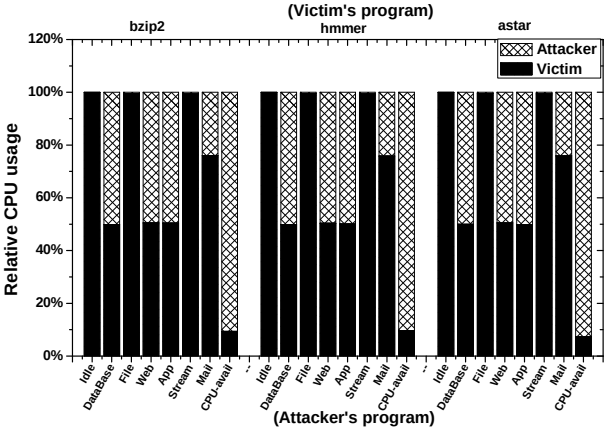

Figure 7: Measurements of CPU Availability Vulnerability.

5. VM Lifecycle and Attestation Responses

Attestations can be performed at all stages of a VM's lifecycle, during VM launch, during its runtime, before and after any VM migrations and on VM termination.

5.1. VM Startup and Responses

Startup attestation can ensure that the VM is correctly initialized and launched. This is an attestation of the integrity of the platform and the VM image. If the platform's integrity is compromised, *CloudMonatt* will select another qualified server for hosting this VM. If the VM image is compromised, then the VM launch request will be rejected. If both the VM and platform pass the integrity checks, the VM will be successfully launched on this server.

5.2. VM Runtime and Responses

CloudMonatt provides a flexible protocol for monitoring the VM's runtime activities, as described in Section 3.4 and Table 1. Customers can issue a one-time attestation request, or a periodic attestation request, during the VM's execution to

monitor its health. *CloudMonatt* provides a set of responses to a VM that is compromised, or under attack. Currently we implement:

#1. Termination: the cloud controller can shut down the VM to protect it from attacks.

#2. Suspension: the controller can temporarily suspend the VM when it detects the platform's security health may be questionable. Meanwhile, it can initiate further checking and also continue to attest the platform. If the attestation results show the cloud server has returned to the desired security health, the controller can resume the VM from the saved state.

#3. Migration: when the security health of the current server is questionable or the server has been compromised, the controller tries to find another secure cloud server that can satisfy the VM's security property requirements. If a suitable server is found, the controller migrates the VM to that server. Otherwise, this VM is terminated for security reasons.

5.3. VM Migration and Responses

A VM may need to migrate to other servers due to resource optimization, or for security reasons. *CloudMonatt* finds a qualified server that supports this VM's security and attestation needs. The VM may need to be shut down if no server is found.

6. Implementation

We implemented our property-based cloud attestation on the OpenStack Havana platform [4]. We integrated the OpenAttestation software (oat) [2] for host remote attestation protocols. We integrated the TPM-emulator [39] and leveraged it to emulate the functions of the *Trust Module* in the hardware. Our evaluation results in Section 7 show that the emulation of the *Trust Module* has little impact on the system performance. Figure 8 displays our prototype implementation.

6.1. Cloud Controller

The Cloud Controller is implemented by the OpenStack Nova. We modify three modules (shown in gray in Figure 8):

nova api: We extend the VM launch command with the monitoring and attestation options: when launching VMs, the customers can specify which properties they want to monitor for their VMs. When the cloud provider searches for a destination machine for initial VM allocation or migration, it must choose servers which support such properties.

Four new commands (Table 1) are added to enable the customers to monitor the VM's health. The customers provide the security properties they want to monitor, the attested VM id, and a nonce, and they will receive the attestation results.

nova database: We modify the controller's database to enable it to store the customers' specifications about the security properties required for their VMs, from *nova api*. We also add new tables in the database, which record each servers' monitoring and attestation capabilities: i.e., what properties they support for monitoring.

nova scheduler: the *nova scheduler* is modified to implement the Policy Validation Module and Deployment Module of the Cloud Controller in Figure 1. It is responsible for choosing the host for the VM during initial allocation and migration. The default scheduler in OpenStack is to choose the server with the most remaining physical resources, to achieve workload balance. We add a new filter: *property_filter*, to select qualified cloud servers to host VMs based on their customers' security properties, monitoring and attestation requirements.

We add two new modules (shown in red) in the controller:

nova attest_service: This essential module manages the attestation services. It connects *nova database* (for retrieving security properties), *oat api* (for issuing attestations and receiving results) and *nova response* (for triggering the responses).

nova response: This implements the Response Module in Figure 1. It is responsible for providing some responses if the attestation fails, as discussed in Section 5.

6.2. Attestation Server

The attestation server and client are realized by OpenAttestation. The Attestation Server has four main modules: *oat database* stores information about the cloud servers and measurements; *oat appraiser* is responsible for triggering attestations and reporting the measurements; *oat PrivacyCA* provides public-key certificates for the cloud servers. We modify *oat api* and add a new module *oat interpreter*:

oat api: We extend the APIs with more parameters, i.e., security properties and VM id.

oat interpreter: This essential new module implements the Property Interpretation and Certification Modules of the Attestation Server. It can interpret the security health of the VM and make attestation decisions, based on the information of the cloud server from the *nova database* and the security measurements from the *oat database*.

6.3. Cloud Servers

In each cloud server, *nova compute* is the client side of OpenStack nova. We modify *oat client*, the client side of OpenAttestation, to receive attestation requests. We modify the *TPM emulator* to provide secure storage and crypto functions. We add two new modules: *Monitor Kernel* can start the security measurements and store the values into the *TPM emulator*, and *Monitor tools* can integrate different software VMI tools, VMM Profile tools or other logging or provenance tools, into the server to perform the monitoring and take measurements.

7. Evaluation

Our testbed includes three Dell PowerEdge R210II servers, each with a quad-core 3.30 GHz Intel Xeon processor, 32GB RAM, and on-board dual Gigabit network adapter with 1 Gbps speed. We select one server as the cloud controller, equipped with Nova Controller and OpenAttestation Server. The other two servers are implemented as cloud server nodes.

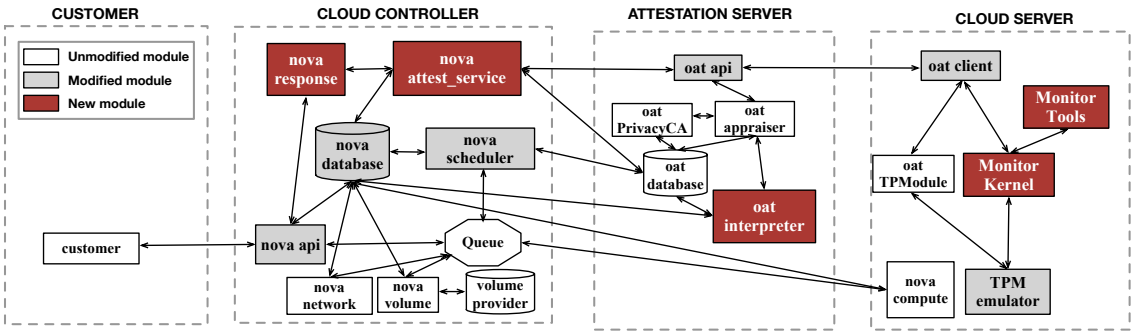

Figure 8: Implementation of Attestation Architecture.

7.1. Performance Evaluation

We consider two performance issues: the overhead of VM launching due to new security requirements, and the overhead of attestation during runtime. We also evaluate different responses for attestation failure recovery. OpenStack Ceilometer [3] is exploited for timing measurements.

7.1.1. VM Launch: In the original OpenStack platform, VM launch involves the following four steps:

- *Scheduling*: allocate VMs to appropriate servers based on customers' requirements and servers' workloads.
- *Networking*: allocate the networks for VMs.
- *Block_device_mapping*: set up block devices for VMs.
- *Spawning*: start VMs on the selected servers.

Our OpenStack *CloudMonatt* architecture involves five steps for VM launching. At the *Scheduling* stage, the controller needs to check *oat database* to find qualified servers which have the security features that support the customer's desired security properties. Steps 2, 3 and 4 are the same as above. We add a fifth stage *Attestation* after the *Spawning* stage. This stage will check if the VM has been launched securely.

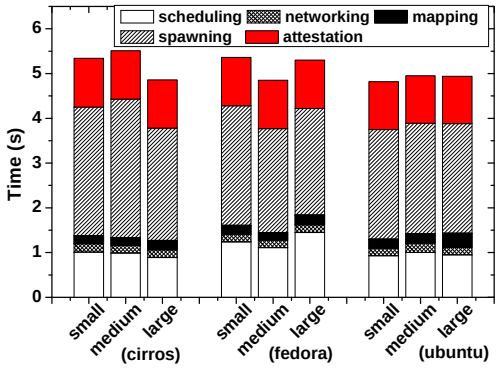

Figure 9: Performance for VM launching.

Figure 9 shows the time for each stage of VM launching. We test three VM images (cirros, fedora and ubuntu) with three VM flavors (small, medium and large). This figure shows that the overhead of the *Attestation* stage is about 20%, which is acceptable for VM launching. The main overhead of an attestation is from the message transmitting in the network.

7.1.2. VM Runtime: During VM runtime, customers can monitor the VM at any time, or periodically at a given frequency. To test the performance effect of periodic runtime attestation, we ran different cloud benchmarks in one virtual machine, while the customer issues the periodic runtime attestation request at different frequencies. Figure 10 shows the effect of periodic runtime attestation at a frequency of 1 minute, 10 seconds and 5 seconds, on ubuntu-large VM.

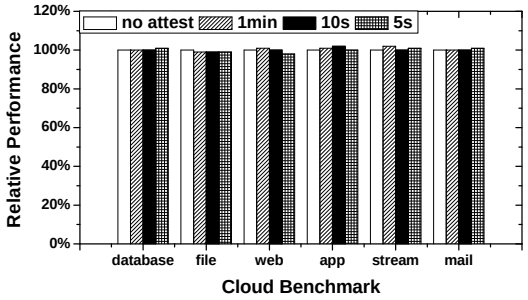

Figure 10: Performance Effect of Runtime Attestation.

This figure indicates that there is no performance degradation due to the execution of runtime attestation. This is for CPU-resource monitoring, where the measurements are taken during the VM switch – the VMM Profile Tool does not intercept the VM's execution. Whether runtime attestation causes performance degradation to the VM execution time depends on the measurement collection mechanism. However, if the periodic attestation frequency is low, then the performance effect is negligible.

7.1.3. Response: The effectiveness of attestation in preventing runtime security breaches depends on two factors: (1) how long it takes to detect potential exploitation of vulnerabilities. This is related to attestation time and mode; and (2) how long it takes to perform the remediation responses. We evaluate the overhead of the defense strategies described in Section 5.

Figure 11 shows the attestation time and reaction time for each response strategy, providing insights into which strategy should be used. Two factors influence the choice of a response: (1) The reaction time of the response should ideally be less than the "damage time", where we define "damage time" as the time from the point at which the attack is detected to the

point at which damage results from the attack. In this respect, Termination is the fastest while Migration is the slowest. (2) The response strategy should also be determined by the specific nature of the attacks and the customers' security needs and usage scenarios. For example, Termination sacrifices VM availability as the customer cannot use the VM any more; Suspension enables the customer to continue the VM only after the server recovers from security breaches; Migration enables the customer to use the VM immediately after the migration is done. So Migration may be the best for service availability.

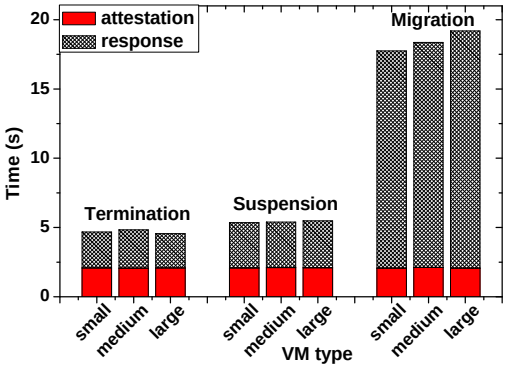

Figure 11: Attestation reaction times during VM runtime.

7.2. Security Evaluation

7.2.1. Server Protection: The trusted entities in the attestation architecture include the Cloud Controller and the Attestation Server. It is important that these machines have secure bootup and are secured for runtime protection. Traditional approaches can be taken to protect these servers, e.g., establishing firewalls, disabling VM launching on these central servers, data hashing and encryption in the database, etc. In addition, the *Trust Module* and *Monitor Module* in the Cloud Servers also need to be protected against hardware or hypervisor attacks via existing protection mechanisms (e.g., [10, 42, 26]).

7.2.2. Protocol Verification: We verify the cryptographic protocol described in Section 3.4 to ensure that customers can receive unforgeable attestation reports.

Protocol properties: We identify several security properties of the protocol for verification:

Secrecy:

① The symmetric keys $\mathbf{K}^x$, $\mathbf{K}^y$, $\mathbf{K}^z$ and the private part of asymmetric keys $\mathbf{SK}^{cust}$, $\mathbf{SK}^c$, $\mathbf{SK}^a$, $\mathbf{SK}^s$, $\mathbf{ASK}^s$ are unknown to the attacker;

② The security properties **P**, measurements **M** and attestation report **R** are unknown to the attacker;

Integrity:

③ The security properties **P**, measurements **M** and attestation report **R** are not modified by the attacker;

Authentication:

④ The customer and Cloud Controller are authenticated and indeed talking with each other;

⑤ The Cloud Controller and Attestation Server are authenticated and indeed talking with each other;

⑥ The Attestation Server and Cloud Server are authenticated and indeed talking with each other.

We use *ProVerif* [9] to verify the above security properties. We model the authentication and communication procedures of our protocol in *ProVerif*, and check the secrecy, integrity and authentication properties defined above.

8. Conclusions

This paper shows how to increase assurance in cloud systems by enabling secure monitoring and attestation of security features provided by a cloud server for the customer's VMs. Key advances over prior work include: (1) Providing a flexible architecture for a rich set of security properties for VM attestation; (2) building the framework for bridging the semantic gap between the security properties a customer wants to request and the measurements collected from a cloud server; (3) enabling initialization as well as runtime attestation during the lifetime of the VM; (4) designing two new cloud-based attacks and the corresponding mechanisms for monitoring those types of confidentiality and availability attacks; (5) defining a novel periodic attestation capability during VM runtime; and (6) building in automated responses to bad attestation results to prevent potential, or further, security breaches. To the best of our knowledge, this is the first real implementation of property-based attestation, for security properties other than integrity checking.

For fast deployability, we leverage existing cloud mechanisms and well-honed security mechanisms where possible, identifying the minimal changes needed for a cloud system to implement our *CloudMonatt* architecture. We also show the set of cryptographic keys that must be present or established, and we define and formally verify our secure attestation protocol. The feasibility of our solution is established by an implementation on the OpenStack cloud software.

We hope that our *CloudMonatt* framework can lay the foundation for future work on monitoring various aspects of security health in cloud computing, and seeing whether these can be seamlessly integrated into *CloudMonatt*. Future work can also lead to further improvements in both the security and the performance of cloud computing.

Acknowledgements

We thank Dr. Pramod Jamkhedkar, now at AT&T Labs, for invaluable help with setting up the OpenStack testbed, and the anonymous reviewers for their feedback on this work. This work was supported in part by the National Science Foundation under grant NSF CNS-1218817. Any opinions, findings, and conclusions or recommendations expressed in this work are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] "Intel trusted execution technology," http://http://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/malware-reduction-general-technology.html/.

[2] "Openattestation project," https://wiki.openstack.org/wiki/OpenAttestation.

[3] "Openstack ceilometer," https://wiki.openstack.org/wiki/Ceilometer.

[4] "Openstack cloud software," http://www.openstack.org/.

[5] "Xen credit scheduler," http://wiki.xen.org/wiki/Credit_Scheduler.

[6] G. I. Apecechea, M. S. Inci, T. Eisenbarth, and B. Sunar, "Fine grain cross-vm attacks on xen and vmware are possible!" *IACR Cryptology ePrint Archive*, 2014.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, 2003.

[8] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, "vtpm: Virtualizing the trusted platform module," in *Proceedings of the Conference on USENIX Security Symposium*, 2006.

[9] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proceedings of the IEEE Workshop on Computer Security Foundations Workshop*, 2001.

[10] D. Champagne and R. Lee, "Scalable architectural support for trusted software," in *Proceedings of the International Symposium on High Performance Computer Architecture*, 2010.

[11] J. Chen and G. Venkataramani, "Cc-hunter: Uncovering covert timing channels on shared processor hardware," in *Proceedings of the IEEE International Symposium on Microarchitecture*, 2014.

[12] L. Chen, R. Landfermann, H. Löhr, M. Rohe, A.-R. Sadeghi, and C. Stüble, "A protocol for property-based attestation," in *Proceedings of the ACM Workshop on Scalable Trusted Computing*.

[13] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware analysis via hardware virtualization extensions," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2008.

[14] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee, "Virtuoso: Narrowing the semantic gap in virtual machine introspection," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2011.

[15] D. Dolev and A. C. Yao, "On the security of public key protocols," Stanford University, Tech. Rep., 1981.

[16] P. England and J. Loeser, "Para-virtualized tpm sharing," in *Proceedings of the International Conference on Trusted Computing and Trust in Information Technologies: Trusted Computing - Challenges and Applications*, 2008.

[17] Y. Fu and Z. Lin, "Space traveling across vm: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.

[18] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing," in *Proceedings of the ACM Symposium on Operating Systems Principles*, 2003.

[19] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proceedings of the Symposium on Network and Distributed Systems*, 2003, pp. 191–206.

[20] T. C. Group, "Tcg software stack specification," http://trustedcomputinggroup.org, Aug. 2003.

[21] T. C. Group, "Design, implementation, and usage principles for tpm-based platforms," May 2005.

[22] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! a fast, cross-vm attack on aes," in *Research in Attacks, Intrusions and Defenses*. Springer, 2014.

[23] T. Jaeger, R. Sailer, and U. Shankar, "Prima: Policy-reduced integrity measurement architecture," in *Proceedings of the ACM Symposium on Access Control Models and Technologies*, 2006.

[24] P. Jamkhedkar, J. Szefer, D. Perez-Botero, T. Zhang, G. Triolo, and R. B. Lee, "A framework for realizing security on demand in cloud computing," in *Proceedings of the IEEE Conference on Cloud Computing Technology and Science*, 2013.

[25] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2007.

[26] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue, and U. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the ACM International Workshop on Hardware and Architectural Support for Security and Privacy*, 2013.

[27] A. Nagarajan, V. Varadharajan, M. Hitchens, and E. Gallery, "Property based attestation and trusted computing: Analysis and challenges," in *Proceedings of the International Conference on Network and System Security*, 2009.

[28] B. Payne, M. Carbone, M. Sharif, and W. Lee, "Lares: An architecture for secure active monitoring using virtualization," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 2008.

[29] D. Perez-Botero, J. Szefer, and R. B. Lee, "Characterizing hypervisor vulnerabilities in cloud computing servers," in *Proceedings of the International Workshop on Security in Cloud Computing*, 2013.

[30] J. Poritz, M. Schunter, E. Van Herreweghen, and M. Waidner, "Property attestation -scalable and privacy-friendly security assessment of peer computers," IBM Research, Tech. Rep., 2004.

[31] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the ACM conference on Computer and communications security*, 2009.

[32] A.-R. Sadeghi and C. Stüble, "Property-based attestation for computing platforms: Caring about properties, not mechanisms," in *Proceedings of the Workshop on New Security Paradigms*, 2004.

[33] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a tcg-based integrity measurement architecture," in *Proceedings of the Conference on USENIX Security Symposium*, 2004.

[34] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu, "Policy-sealed data: A new abstraction for building trusted cloud services," in *Proceedings of the Conference on USENIX Security Symposium*, 2012.

[35] V. Scarlata, C. Rozas, M. Wiseman, D. Grawrock, and C. Vishik, "Tpm virtualization: Building a general framework," in *Trusted Computing*. Vieweg+Teubner, 2008.

[36] J. Schiffman, T. Moyer, H. Vijayakumar, T. Jaeger, and P. McDaniel, "Seeding clouds with trust anchors," in *Proceedings of the ACM Workshop on Cloud Computing Security Workshop*, 2010.

[37] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," in *Proceedings of the ACM Symposium on Operating Systems Principles*, 2005.

[38] E. Shi, A. Perrig, and L. van Doorn, "Bind: a fine-grained attestation service for secure distributed systems," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2005.

[39] M. Strasser and H. Stamer, "A software-based trusted platform module emulator," in *Trusted Computing-Challenges and Applications*. Springer, 2008.

[40] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, "Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense)," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2012.

[41] M. Velten and F. Stumpf, "Secure and privacy-aware multiplexing of hardware-protected tpm integrity measurements among virtual machines," in *Proceedings of the International Conference on Information Security and Cryptology*, 2013.

[42] Z. Wang and X. Jiang, "Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2010.

[43] M. Weiß, B. Heinz, and F. Stumpf, "A cache timing attack on aes in virtualization environments," in *Financial Cryptography and Data Security*. Springer, 2012.

[44] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: High-speed covert channel attacks in the cloud." in *Proceedings of the Conference on USENIX Security Symposium*, 2012.

[45] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting, "An exploration of l2 cache covert channels in virtualized environments," in *Proceedings of the ACM workshop on Cloud computing security workshop*, 2011.

[46] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," in *Proceedings of the ACM conference on Computer and communications security*, 2012.

[47] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-tenant side-channel attacks in paas clouds," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2014.

[48] F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram, "Scheduler vulnerabilities and coordinated attacks in cloud computing," in *Proceedings of the IEEE International Symposium on Network Computing and Applications*, 2011.